

Channel Decoding “Soft” Acceleration on the TigerSHARC DSP



Introduction

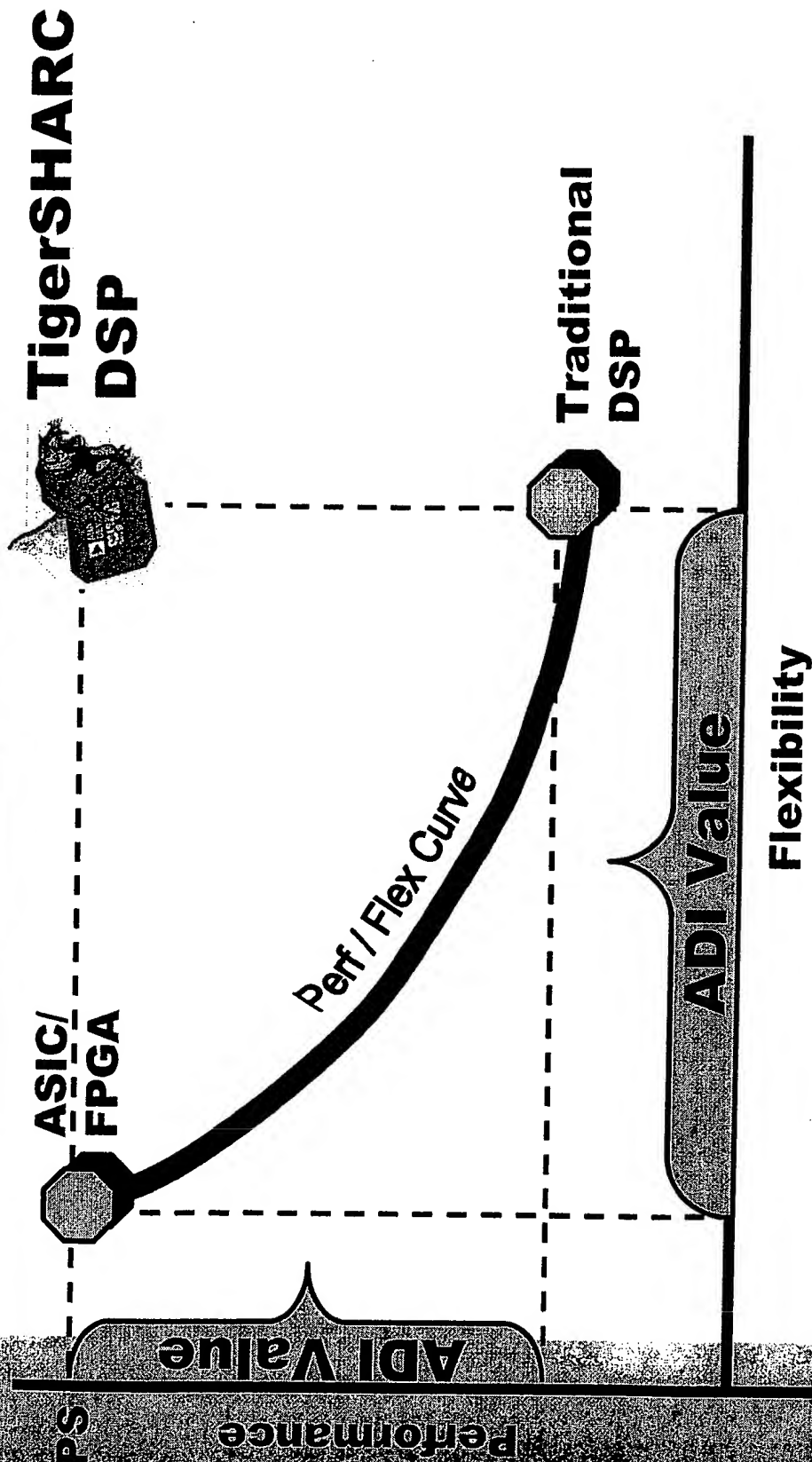
- The TigerSHARC DSP is a balance of many parallel resources and high bus bandwidth.
- The parallel resources offer high computation rates at moderate clock speeds.
- The high bus bandwidth enables high data rate processing.
- Channel decoding algorithms, Viterbi and turbo decoders, contain parallel characteristics allowing the processing to be split across many resources.
- The TigerSHARC architecture is highly suited to channel decoder processing.

Received Signal Distribution $\sigma=0.5$

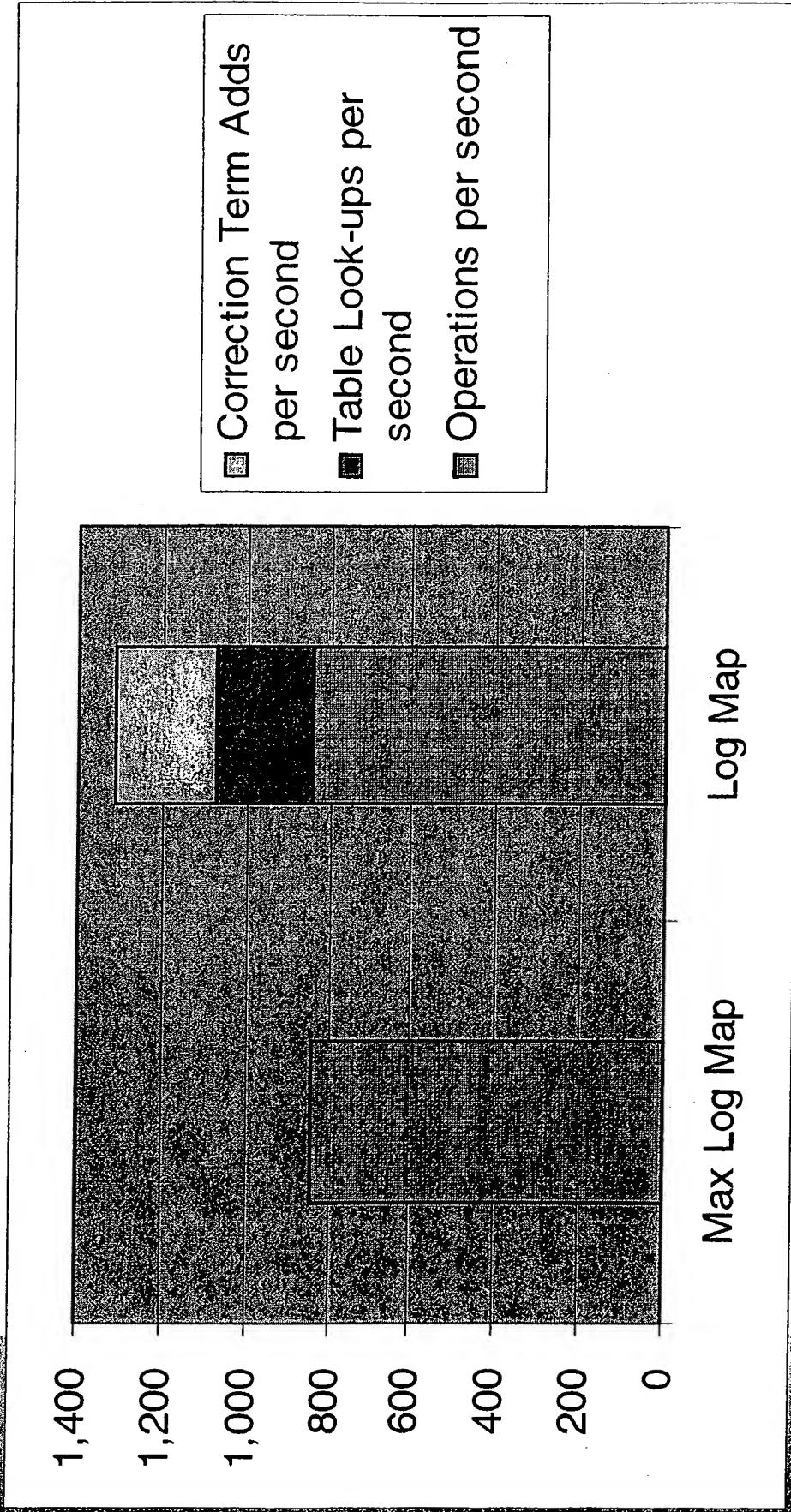
Received Signal Distribution $\sigma=1.5$

7 bits in (3.4), 6 bits out

Changing the Performance / Flexibility Curve for next gen base station systems



Complexity Comparison of Max-Log-Map to Log-Map



384 kbps, 8 iterations, K=4

TigerSHARC Soft Acceleration Performance

8-Bit Viterbi	N*91	N*67	
	N*23	N*23	
	N*52	N*28	
	N*9	N*9	
	N*7	N*7	
16-Bit Viterbi	N*128	N*80	
	N*17	N*17	
	N*104	N*56	
	N*7	N*7	
MAP Decoder	N*45.50	N*10.0	N*10.0
	N*0.50	N*0.50	N*0.50
	N*24.50	N*4.00	N*4.00
	N*20.50	N*5.50	N*5.50

TigerSHARC Acceleration Approach

- Hardware support for core components of Channel Decoding (trellis butterfly)
 - Single instruction to Add, Compare and Select (ACS)
- Additional hardware support for optimal Turbo decoder solution (Log MAP vs MAX Log MAP)
 - Single instruction (TMAX) for calculation of $\ln(e^a + e^b)$
- Parallel resources for execution of channel decoder algorithms
- Extended precision support for accumulation of 16/8-bit inputs
- High internal bus bandwidth

Channel Decoding Algorithms

■ Viterbi

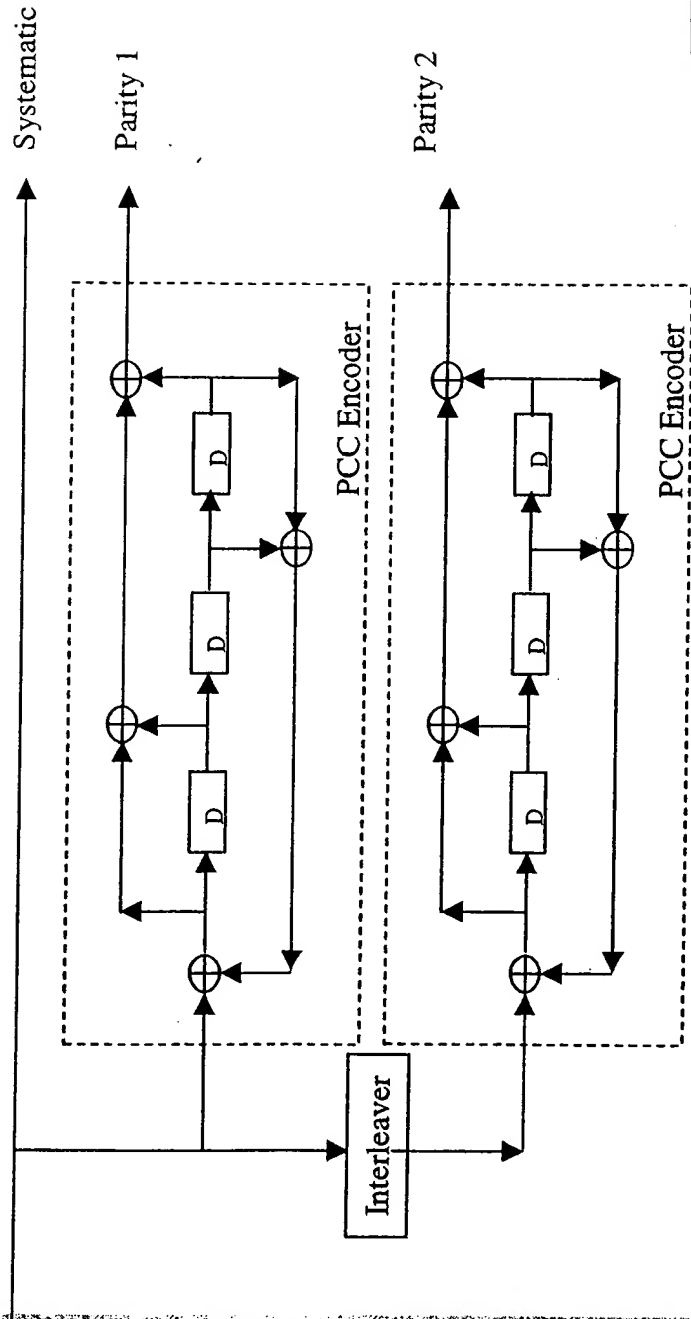
- Channel Decoder for Convolutional codes
- Data Rates at or below 32kbps
- Maximum Likelihood Algorithm (ML)

■ Turbo

- Iterative Channel Decoder for Parallel Concatenated Convolutional Coding (PCCC)
- Data Rates at or above 32kbps
- Maximum A Posteriori Algorithm (MAP)

Parallel Concatenated Convolutional Coding (PCCC)

- Inputs
- Original data (Bit stream)
- Polynomials
- Interleave Index
- Outputs
- Original data $u(k)$
- Parity bits $\mathbf{P}(k) = [P_1(k), P_2(k), \dots, P_N(k)]$



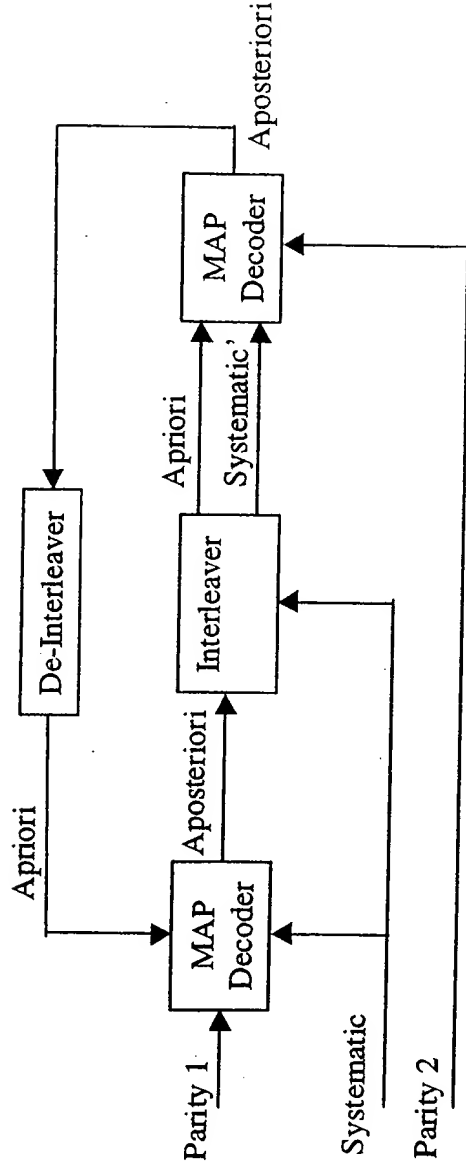
Turbo Decoding: Maximum A Posteriori (MAP)

■ Inputs

- ☐ Received Systematic data $y^u(k)$
- ☐ Received Parity bits $y^{p_N}(k)$
- ☐ Polynomials
- ☐ Interleave Index
- ☐ Number of iterations/
stopping algorithm

■ Outputs

- ☐ Original data (Bit stream)
- ☐ Likelihood Ratio (LLR)
- ☐ Extrinsic Information



Maximum A Posteriori Decoder: Log Likelihood Ratio

$$LLR(k) = \ln \left(\frac{P(u_k = 1 | \vec{y})}{P(u_k = 0 | \vec{y})} \right)$$

Where

$$P(u_k = x | \vec{y}) = \sum_{s^x} P(s_{k-1} = s', s_k = s, \vec{y}) / P(\vec{y})$$

and

$$P(s_{k-1} = s', s_k = s, \vec{y}) = \alpha(s_{k-1}) \gamma(s', s) \beta(s_k)$$

Maximum A Posteriori (MAP) Decoder: Metric Calculation

$$\alpha_k(s) = P[s, (y_1, \dots, y_k)] = \sum_{s \in S} \alpha_{k-1}(s') \gamma_k(s', s)$$

$$\beta_{k-1}(s') = P[(y_k, \dots, y_N) | s'] = \sum_{s \in S} \beta_k(s) \gamma_k(s', s)$$

$$\gamma_k(s', s) = P[s_k = s, y_k \mid s_{k-1} = s'] = P[u_k] P[y_k | u_k]$$

Maximum A Posteriori (MAP) Decoder: Gamma Metric

$$\gamma_k(s', s) \sim e^{[u_k(L_k^e + L_c y_k^s) + L_c y_k^p x_k^p]/2}$$

- u_k -> input (-1,1)
- x_k^p -> generated parity (-1,1)
- L_k^e -> Apriori Information
- L_c -> Noise parameter (
- y_k^s -> Systematic Input (soft?? -1,1)
- y_k^p -> received parity (soft?? -1,1)

Maximum A Posteriori (MAP) Decoder: alpha metric Calculation in Log Domain

$$\begin{aligned}\ln[\alpha_k(s)] &= \ln[\sum \alpha_{k-1}(s') \gamma_k(s', s)] \\ &= \ln[\sum \alpha_{k-1}(s') e^{[u_k(L_k^e + L_c y_k^s) + L_c y_k^p x_k^p]/2}]\end{aligned}$$

Using Jacobian Logarithm:

$$\begin{aligned}\ln[\alpha_k(s)] &= \text{MAX}\{ \ln[\alpha_{k-1}(s')] + \ln[\gamma_k(s', s)] , \\ &\quad \ln[\alpha_{k-1}(s'')] - \ln[\gamma_k(s', s)] \} \\ &\quad + \ln[1 + e^{-\text{abs}(\ln[\alpha_{k-1}(s')] - \ln[\alpha_{k-1}(s'')])}]\end{aligned}$$

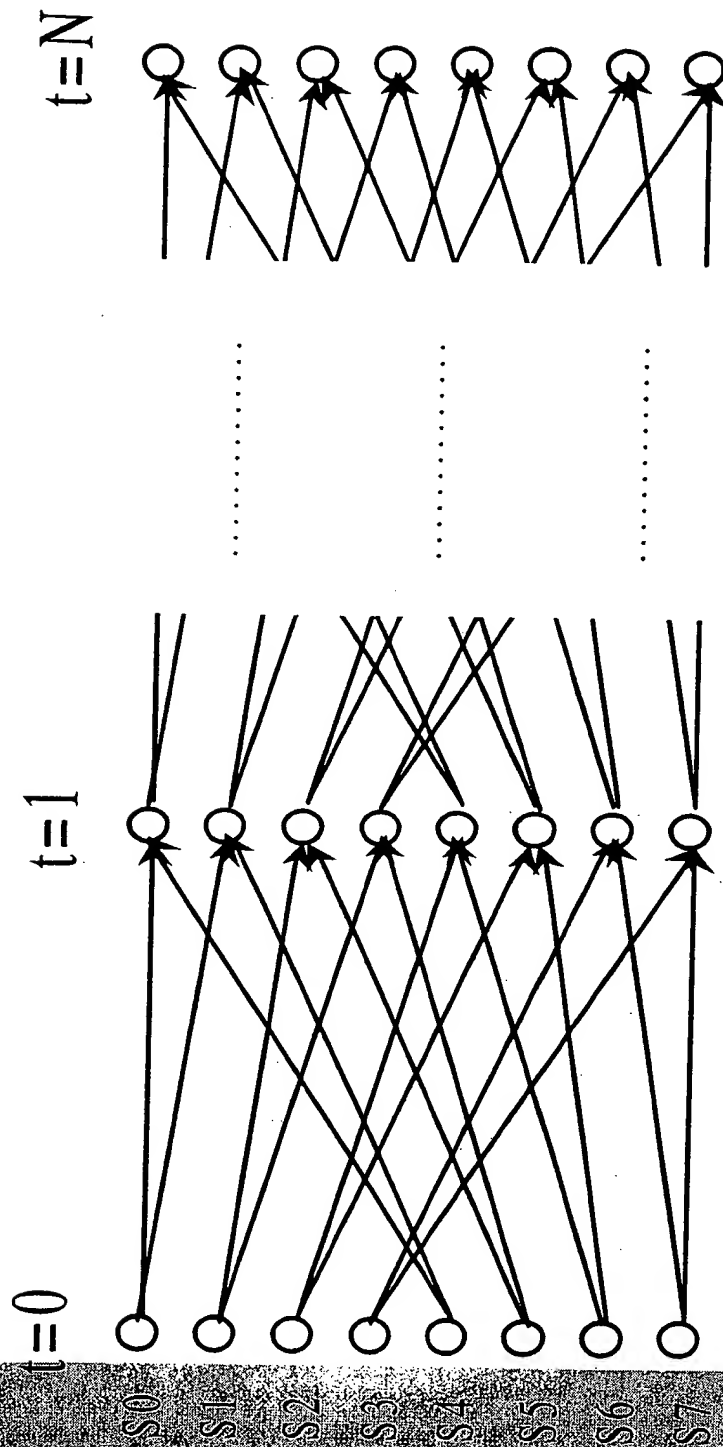
Maximum A Posteriori (MAP) Decoder: Beta Metric Calculation in Log Domain

$$\begin{aligned} [\beta_{k-1}(s)] &= \text{Ln}[\sum \beta_k(s') \gamma_k(s', s)] \\ &= \text{Ln}[\sum \beta_k(s') e^{[u_k(L_k^e + L_c y_k^s) + L_c y_k^p x_k^p]/2}] \end{aligned}$$

Using Jacobian Logarithm:

$$\begin{aligned} [\beta_{k-1}(s)] &= \text{MAX}\{ \text{Ln}[\beta_k(s')] + \text{Ln}[\gamma_k(s', s)] , \\ &\quad \text{Ln}[\beta_k(s'')] - \text{Ln}[\gamma_k(s', s)] \} \\ &\quad + \text{Ln}[1 + e^{-\text{abs}(\text{Ln}[\beta_k(s')] - \text{Ln}[\beta_k(s'')])}] \end{aligned}$$

Trellis: Viterbi and Turbo Decoders



Log MAP Decoder Implementation: Alpha and Beta

- Calculate Alpha and Beta metrics via Trellis
 - Log MAP: Include look up for correction term (precision?? >5 0, output < 0.7)
 - MAX Log MAP: Ignore look-up for correction term (0.5dB Loss)
- Initialization of Alpha and Beta
 - $\text{Ln}(1)$ for state 0 at $k=0$ for a and $k=n$ for b
 - $\text{Ln}(0)$, $-\text{inf}$, for all other states
- Precision and scaling during accumulation in Trellis
 - ACS instruction provides 32-bit precision
 - Processing increases if 16-bit precision used

Trellis Butterfly Instruction

■ ACS Instruction

■ $TRsq = ACS(TRmd, TRnd, Rm)$ (TMAX)

■ Executes the functionality of a Trellis butterfly

- Adds and subtracts Rm to $TRmd$ and $TRnd$
- Selects maximum and adds correction factor
- Places in correct order for next time series calculation

■ Performs 2 butterflies maintaining 32 bit precision in a single cycle and in 1 instruction

■ ACS instruction can be executed SIMD style in both compute blocks

Log MAP Decoder Implementation: Alpha and Beta Memory vs Speed

■ Priority: Speed

- Calculate Gamma Metric
- Calculate Alpha and Beta in Parallel
- Calculate Extrinsic

■ Priority: Memory Utilization

- Calculate Gamma and Alpha metric
- Calculate Beta and Extrinsic

Alpha and Beta Calculation Code: Trellis Butterfly

8 Butterflies in 2 Cycles

4 in X (Alpha calculation)

4 in Y (Beta calculation)

4 Time Samples

```

TR11:8 = ACS(TR5:4, TR1:0, sR24); q[K22+=4]=xR3:0; q[J22-=4]=yR3:0;;
TR15:12 = ACS(TR7:6, TR3:2, sR25); q[K22+=4]=xR7:4; q[J22-=4]=yR7:4;;

TR11:8 = ACS(TR13:12, TR9:8, sR26); q[K22+=4]=xR11:8; q[J22-=4]=yR11:8;;
TR7:4 = ACS(TR15:14, TR11:10, sR27); q[K22+=4]=xR15:12; q[J22-=4]=yR15:12;;

TR11:8 = ACS(TR5:4, TR1:0, sR28); q[K22+=4]=xR3:0; q[J22-=4]=yR3:0;;
TR15:12 = ACS(TR7:6, TR3:2, sR29); q[K22+=4]=xR7:4; q[J22-=4]=yR7:4;;

TR11:8 = ACS(TR13:12, TR9:8, sR30); q[K22+=4]=xR11:8; q[J22-=4]=yR11:8;;
TR7:4 = ACS(TR15:14, TR11:10, sR31); q[K22+=4]=xR15:12; q[J22-=4]=yR15:12;;
if C0E, jump loop;;

```



Memory Requirements of Log-Map Turbo Decoder

Log Likelihood Ratio Calculation

$$\text{LLR}(k) = \ln \left(\frac{P(u_k = 1 | \hat{\mathbf{y}})}{P(u_k = 0 | \hat{\mathbf{y}})} \right)$$

or

$$\text{LLR}(k) = \ln \left(\frac{\sum_{\mathbf{s}^+} \alpha(\mathbf{s}_{k-1}) \gamma(\mathbf{s}', \mathbf{s}) \beta(\mathbf{s}_k)}{\sum_{\mathbf{s}^-} \alpha(\mathbf{s}_{k-1}) \gamma(\mathbf{s}', \mathbf{s}) \beta(\mathbf{s}_k)} \right)$$

Log Likelihood Ratio Calculation Using Jacobian Logarithm

$$\begin{aligned} -R(k) = & \text{Max}_{S^+} \{ \ln[\alpha(s_{k-1})] + \ln[\gamma(s',s)] + \ln[\beta(s_k)] \} \\ & - \text{Max}_{S^-} \{ \ln[\alpha(s_{k-1})] + \ln[\gamma(s',s)] + \ln[\beta(s_k)] \} \\ & + \text{Jacobian Correction Factor} \end{aligned}$$

Jacobian Logarithm Instruction for LLR Calculation

■ TMAX Instruction

□ $TRsd = TMAX(TRmd + Rmq_h, TRnd + Rmq_l)$

□ Executes the functionality of a Jacobian Logarithm

- Adds the value of two metrics ($\ln(\text{Alpha}) + \ln(\text{Beta})$)
- Selects maximum between the two summations
- Adds correction factor

□ Can perform operation on two time samples in parallel in 1 instruction

■ TMAX instruction can be executed SIMD style in both compute blocks

Log Likelihood Ratio Calculation Code

```
//ADD Alpha + Beta's for 0 and 1 state changes
TR9:8 = TMAX(TR1:0 + R9:8, TR3:2 + R11:10);;
TR11:0 = TMAX(TR1:0 + R13:12, TR3:2 + R15:14);;
TR13:12 = TMAX(TR5:4 + R13:12, TR7:6 + R15:14);;
TR15:14 = TMAX(TR5:4 + R9:8, TR7:6 + R11:10);;
```

```
//Add Gamma for 0 and 1 state changes
```

```
TR5:4 = TMAX(TR9:8 + R0:0, TR13:12 + R1:1);;
TR12:11 = TMAX(TR11:10 - R0:0, TR15:14 - R1:1);;

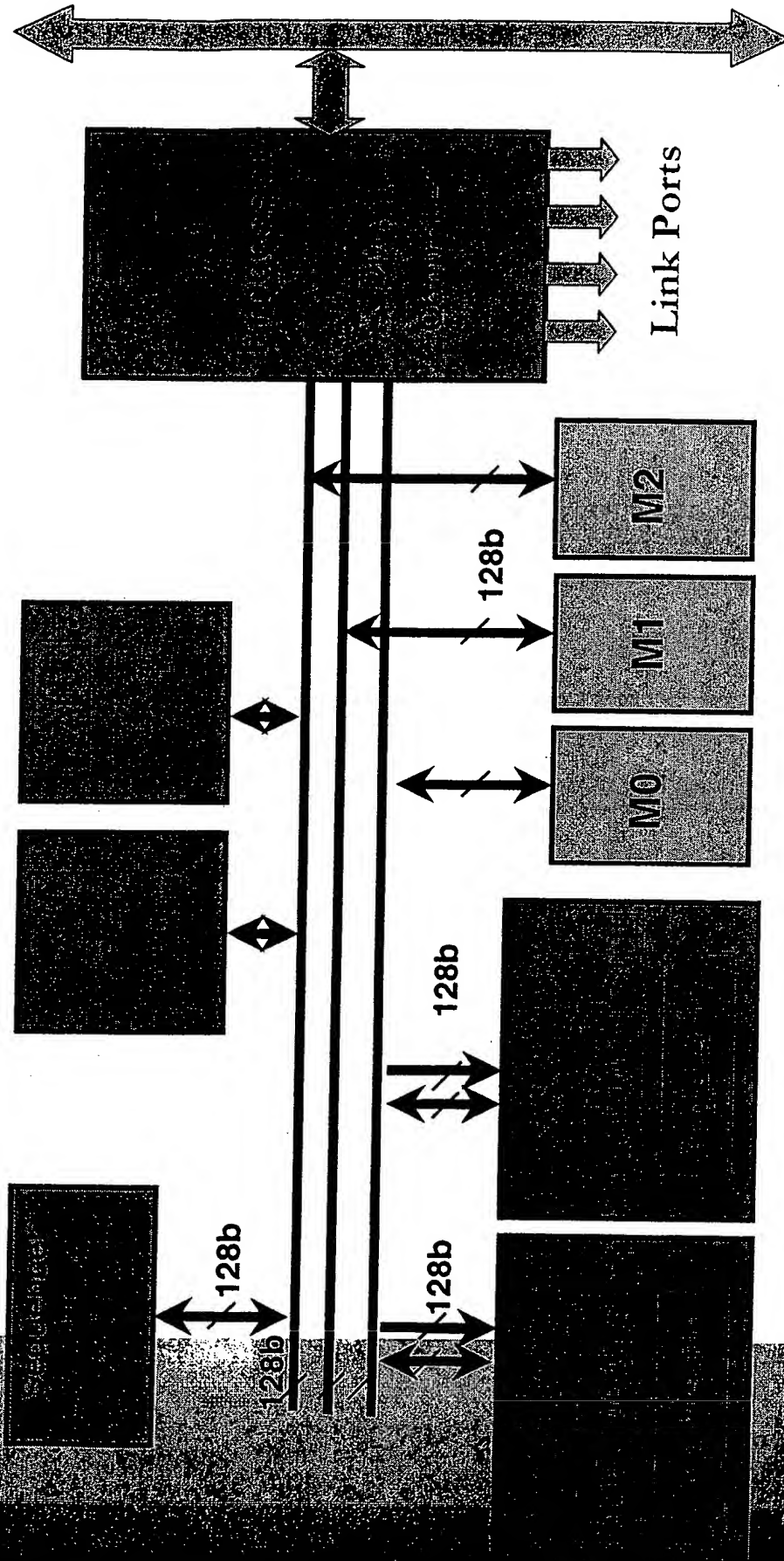
R0 = TMAX(TR5, TR4);;
R1 = TMAX(TR12, TR11);;

R0 = R1 - R0;;
```

TigerSHARC Solution

- Many parallel resources
 - 1 instruction can deliver multiple operations
 - Parallelism can be effectively used on trellis butterfly operations
- Large Internal Memory
 - 6 Mbit version available today
 - Large Mbit versions based on eDRAM technology
- Very High Internal Memory Bandwidth
 - 32 bytes per cycle

A New Class of DSP: TigerSHARC



2 Computation blocks X and Y

External Bus



Forward Error Correction Soft Acceleration for TigerSHARC

- What: Add three new instructions to reduce TigerSHARC Clock cycle requirements

- ☐ TMAX

- ☐ ACS

- ☐ PERMUTE

- Why: Eliminate the need for an ASIC solution for channel decoding

Topics of Special Interest

- Soft input data quantization levels
 - Systematic and Parity
 - Apriori
- Error Correction table quantization levels
- Scaling of extrinsic for conversion to apriori
 - Extrinsic calculation from LLR
- Gamma equation
- Alpha and Beta initial values (precision during accumulation)
- Number of iterations expected
- IP